



Security Requirements Engineering Methodologies

***Nicola Zannone, Fabio Massacci
and John Mylopoulos***

***Department of Information and Communication Technology
University of Trento***



Talk Outline

- Security Requirements Engineering
- Reviewing the state-of-the-art
- Secure Tropos
 - ✓ Modeling Framework
 - ✓ Formal Framework
- Computer-Aided SRE
- Ongoing and Future Work



Security and Software Engineering

- Traditional SE approaches treat security as a non-functional requirement
- This introduce quality constraints under which the system must operate
- Need to integrate non-functional requirements (e.g., reliability and performance) into the software development process
- Security still remains an afterthought



Security and Requirements

- The usual approach towards the inclusion of security within a system is to identify security requirements **after** system design
- Security mechanisms have to be fitted into a pre-existing design
 - ✓ may not be able to accommodate them
 - ✓ security requirements can generate conflicts with functional requirements of the system
- **Big gap** between solutions and the requirements of the entire system



Security vs Software Engineering

- **Software Engineer:** design a system so that legitimate users can do what they want to do
- **Security Engineer:** design a system so that illegitimate users cannot do what they should not do
- **Contentious Consequence**
 - ✓ We cannot use traditional Requirements or Software Engineering methodologies for Security, they have different overall goals!



Some Discarded Ideas...

- **Discarded idea 1**
 - ✓ Add primitives/constraints to Tropos/Kaos/UML/name-your-pet-RE-formalism to accommodate various security requirements
 - ✓ Confidentiality, authentication, access controls, etc., are security services and mechanisms NOT security requirements!
 - ✓ ACID Transactions are a DB service not an IS requirement...
- **Discarded idea 2**
 - ✓ Model security requirements separately from functional requirements
 - ✓ Well, where's the distinction then? Why bother?
- **Discarded Idea 3**
 - ✓ Model the goals of the attacker
 - ✓ They are not the goals of the security engineer!



Other Ideas

→ UML Proposals

- ✓ SecureUML, Model-Driven Architecture [Basin et al.]
- ✓ UMLsec - [Juriens]
- ✓ Abuse-Cases [McDermott & Fox]
- ✓ Misuse Cases [Sindre & Opdhal]

→ Early Requirements Proposals

- ✓ Anti-requirements [van Lamsweerde et al., Crook et al.],
- ✓ Problem-Frames, Abuse Frames [Hall et al., Lin et al]
- ✓ Security Patterns [Giorgini & Mouratidis]
- ✓ Privacy Modelling [Liu et al., Anton et al.]



Early Discussion

→ UML Pros and Cons

- ✓ Well-known even if meta-level extension not standardized
- ✓ Effective - MDA -> automatic configuration of system
- ✓ "Too Late" - model of system rather than organization

→ Early requirements Pros and Cons

- ✓ Capture organizational structure
- ✓ Modelling done at object-level (limited extensions)
- ✓ "Too Functional" - Security is modelled explicitly and in parallel with the actual functional model



Object- vs Meta-Level Approaches

→ Object-level approach

- ✓ Use an off-the-shelf framework (e.g., UML, Kaos, i*/Tropos) as it is for modeling security requirements
 - ✓ Pro: modeling well established, reasoning feature ready
 - ✓ Con: modeling often cumbersome, some time impossible

→ Meta-level approach

- ✓ Take a RE framework and enhance it with novel constructs specific to security
 - ✓ Pro: modeling more effective and compact
 - ✓ Con: language constructs must gain “market acceptance”, semantics and reasoning to be update



Security Requirements Engineering

→ Most proposals focus on

- ✓ protection aspects of security and explicitly deal with a series of security service (integrity, availability, etc.)
- ✓ related protection mechanisms (password, crypto, etc.)

→ Our goal is to introduce security requirements analysis in the early phases of the software development process

→ This allows us to

- ✓ elicit security requirements from the organizational environment
- ✓ analyze security requirements within the organizational environment in which the software will operate
- ✓ motivate the use of specific security mechanisms

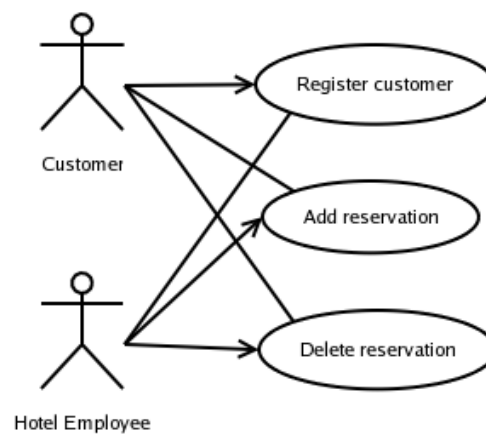


Use Case Diagram

- Build a first sketch model of a system
- Characterize a way of using a system
- Offer a notation for describe the functionality of a system
 - ✓ Actors: an abstraction of an external agent that interact with the system
 - ✓ Use cases: specification of a type of interaction between a system and agents
 - ✓ Association lines: connect agents with the use cases in which they participate



Reservation System



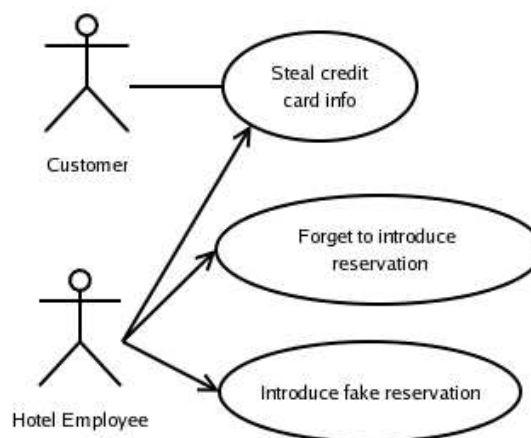


Abuse Cases

- McDermott & Fox, 1999
- Negative use cases for modeling security requirements
- Specify an interaction between a system and one or more actors, where the results of the interaction are harmful to the system or one of the actors in the system
- Actors are the same that participate in use cases



Reservation System





Limits of Abuse Cases

- **Model security requirements separately from functional requirements**
 - ✓ Abuse case diagrams show abuse only, not abuse together with normal use
 - ✓ They do not investigate relations between use and abuse



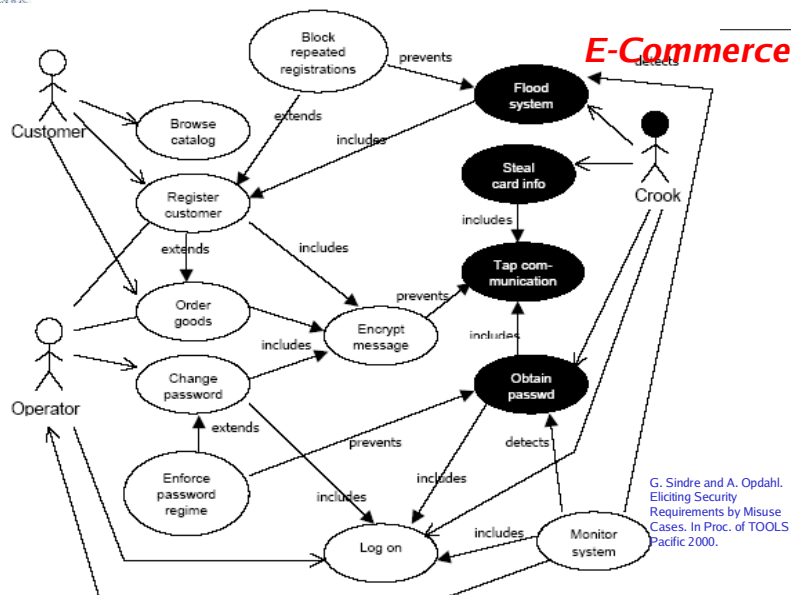
Misuse Case

- Guttorm Sindre and Andreas Opdahl, 2000
- Extend use cases for modeling security requirements
- Specify behavior that the system should avoid
- Specify how a misuser can damage the system



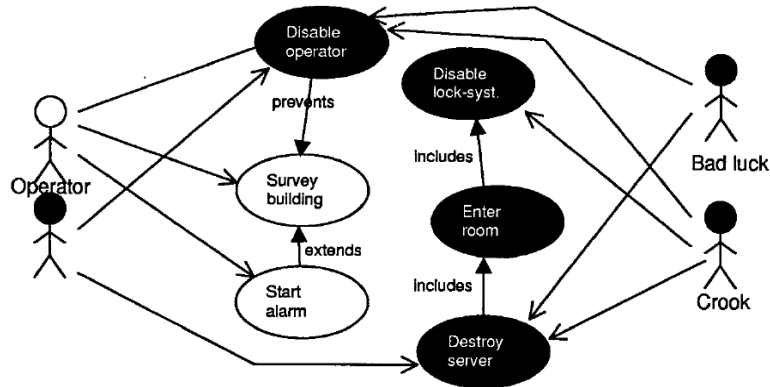
Concepts

- **Misuser**
 - ✓ hostile actor
 - ✓ a similar notation as an actor in use cases, except the misuser has a black "head" instead of white
- **Misuse case**
 - ✓ course of actions performed to do harm to a stakeholder or the system itself
 - ✓ behavior that is not wanted in the system
 - ✓ illustrated by black circles
- **Use cases**
 - ✓ functionalities of the system
 - ✓ countermeasures against misuse
- **Relations**
 - ✓ "includes" and "extends"
 - ✓ "prevents": use case prevents the activation of a misuse case
 - ✓ "detects": use case detects the activation of a misuse case





Special mis-actors



G. Sindre and A. Opdahl. Eliciting Security Requirements by Misuse Cases. In Proc. of TOOLS Pacific 2000.

@2006 Zannone, Massacci, Mylopoulos

Secure Tropos -- 19



Advantages

- Focus on security in the early phases of the software development process
- Increase the chance of discovering threats that otherwise would have been ignored
- Help to trace and organize the requirements specification
- Help to evaluate requirements
 - ✓ the real cost of implementing a use case includes the protection needed to mitigate all serious threats to it
- Easy to reuse in new development projects

@2006 Zannone, Massacci, Mylopoulos

Secure Tropos -- 20



Disadvantages

- Use/Misuse case are informal
 - ✓ No clear semantics
 - ✓ (Hence) NO formal analysis
- No knowledge on how to write *good quality* misuse cases
- The focus is **ONLY** on the system-to-be
- NOT suitable for all kind of threats
- There is not always an identifiable misuser and the misuse case may not always consist of an identifiable sequence of actions



KAOS

- A research project
- Used to formalize goals into requirements
- Derive a description of a system's behavior
- Analyze system structure through acquiring and formalizing functional and non-functional requirements
- Supported by GRAIL tool



KAOS concepts

- **Agents**
 - ✓ active component of the system
 - ✓ play some role
- **Goals**
 - ✓ prescriptive statements of intent about the system
 - ✓ functional goals: service to be provided
 - ✓ non-functional goals: quality of service
- **Domain properties**
 - ✓ descriptive statements about the environment (e.g., physical laws, norms)



Goals

- **Organized in terms of AND/OR hierarchies**
 - ✓ Goal refinement used to construct a refinement-abstraction hierarchy
 - ✓ High level goals are strategic
 - ✓ Coarse grained with many agents
 - ✓ Low level goals are technical
 - ✓ Fine grained involving less agents
- **Requirement: terminal goal for one agent**



Obstacles

- Identify goal violation scenarios
- An obstacle to some goal is a condition whose satisfaction may prevent the goal from being achieved
- An obstacle O is said to obstruct a goal G in domain Dom iff

$\{O, Dom\} \models \neg G$ Obstruction

$Dom \models \neg O$ Domain consistency



Obstacles Analysis

- Obstacles analysis consists in taking a pessimistic view of goals
- Identify as many ways of breaking goals as possible in order to resolve each such situation
- Formal techniques for generation and AND/OR refinement of obstacles are available
- Obstacles need to be resolved once they have been generated
 - ✓ Resolution techniques: goal substitution, agent substitution, goal weakening, goal restoration, obstacle prevention and obstacle mitigation
- Obstacle analysis is a recursive process
 - ✓ it may produce new goals for which new obstacles may be generated and resolve



Security Goals

- Considered a meta-class
- High level of abstraction
 - ✓ Confidentiality
 - ✓ Integrity
 - ✓ Availability
 - ✓ Privacy
 - ✓ Authenticity
 - ✓ Non-repudiation
- Each security goal has to be instantiated into application-specific security goal



Confidentiality

Goal Confidentiality



Confidentiality

Goal Confidentiality

Goal *Avoid*[SensitiveInfoKnownByUnauthorizedAgent]

FormalSpec $\forall$ ag: Agent, ob: Object

$\neg \text{Authorized}(\text{ag}, \text{ob}, \text{Info}) \Rightarrow \neg \text{Knows}(\text{ob}, \text{Info})$

If an agent **ag** is not authorized to access info about an object **ob**, then he does not know any information info about the object **ob**



Confidentiality

Goal Confidentiality

Goal *Avoid*[SensitiveInfoKnownByUnauthorizedAgent]

FormalSpec $\forall$ ag: Agent, ob: Object

$\neg \text{Authorized}(\text{ag}, \text{ob}, \text{Info}) \Rightarrow \neg \text{Knows}(\text{ob}, \text{Info})$

Goal *Avoid*[PaymentMediumKnownBy3rdParty]

FormalSpec $\forall$ p: Agent, acc: Account

$\neg (\text{Owns}(\text{p}, \text{acc}) \vee \text{Manages}(\text{p}, \text{acc}))$

$\Rightarrow \neg (\text{Knows}(\text{acc}, \text{Acc\#}) \wedge \text{Knows}(\text{acc}, \text{PIN}))$

*If agent **p** is not the owner of account **acc** and he should not manage it, he does not know **number** and **PIN** of the account*



Obstacles Analysis

→ Taking the negation of the goal

$\forall p: \text{Agent}, \text{acc}: \text{Account}$
(NG) $\neg (\text{Owns}(p, \text{acc}) \vee \text{Manages}(p, \text{acc}))$
 $\wedge (\text{Knows}(\text{acc}, \text{Acc\#}) \wedge \text{Knows}(\text{acc}, \text{PIN}))$

→ Suppose that the domain theory contains the following properties

(D1) $\forall p: \text{Agent}, \text{acc}: \text{Account} \text{ Owns}(p, \text{acc}) \wedge \text{Knows}(p, \text{name}) \Rightarrow \text{Knows}(\text{acc}, \text{Acc\#})$
(D2) $\forall \text{acc}: \text{Account} \text{ Knows}(\text{acc}, \text{Acc\#}) \Rightarrow \text{Knows}(\text{acc}, \text{PIN})$

→ We can formally derived the following potential obstacle

(O) $\forall p: \text{Agent}, \text{acc}: \text{Account}$
 $\neg (\text{Owns}(p, \text{acc}) \vee \text{Manages}(p, \text{acc})) \wedge \text{Knows}(p, \text{name})$



Anti-goals

- Obstacles sufficient for modeling and resolving non-intentional obstacles (accidental obstacles)
- Too limited for modeling and resolving intentional obstacles (malicious obstacles)
- Active attackers be modeled as well together with their own goals, capabilities, and the vulnerabilities they can monitor or control (anti-models)
- Anti-goals are the intentional obstacles to security goals



Building Anti-models

- Root anti-goal are obtained by negation of security goals
- For each anti-goal, potential attacker are identified (WHO)
- For each anti-goal and corresponding attacker, the higher level anti-goals are identified (WHY)
- For each anti-goal and corresponding attacker, the lower level anti-goals are identified (HOW)
- AND/OR refinement process for anti-goals
 - ✓ realizable by the attacker (anti-requirements)
 - ✓ realizable by the attackee (vulnerabilities)
- Anti-models are derived from anti-goals formulations
- Anti-requirements are defined in terms of the capabilities of the corresponding attacker



Running Antigoals

Goal *Avoid*[PaymentMediumKnownBy3rdParty]

FormalSpec $\forall p: \text{Agent}, \text{acc}: \text{Account}$

$\neg (\text{Owns}(p, \text{acc}) \vee \text{Manages}(p, \text{acc}))$

$\Rightarrow \neg (\text{Knows}(\text{acc}, \text{Acc\#}) \wedge \text{Knows}(\text{acc}, \text{PIN}))$

AntiGoal *Achieve*[PaymentMediumKnownBy3rdParty]

FormalSpec $\exists ag: \text{Agent}, ob: \text{Object}$

$\neg (\text{Owns}(p, \text{acc}) \vee \text{Manages}(p, \text{acc}))$

$\wedge \text{Knows}(\text{acc}, \text{Acc\#}) \wedge \text{Knows}(\text{acc}, \text{PIN})$



Refining antagoals

By asking “what are sufficient conditions for someone unauthorized to know the number and PIN of an account simultaneously?”

AntiGoal *Achieve*[PaymentMediumKnownBy3rdPartyFromPinSearching]

FormalSpec $\exists$ ag: Agent, ob: Object

$\neg (\text{Owns}(p, \text{acc}) \vee \text{Manages}(p, \text{acc})) \wedge \text{Knows}(\text{acc}, \text{Acc\#})$

$\wedge (\exists x:\text{PIN}) [\text{Find}(p, x) \wedge \text{Match}(x, \text{acc}, \text{Acc\#})]$

AntiGoal

Achieve[PaymentMediumKnownBy3rdPartyFromAccountNumberSearching]

FormalSpec $\exists$ ag: Agent, ob: Object

$\neg (\text{Owns}(p, \text{acc}) \vee \text{Manages}(p, \text{acc})) \wedge \text{Knows}(\text{acc}, \text{PIN})$

$\wedge (\exists y:\text{Acc\#}) [\text{Find}(p, y) \wedge \text{Match}(\text{acc}, \text{PIN}, y)]$



Limits of Antagoals

- Modeling attackers is difficult
- We have to consider all the possible obstacles even the ones unknown
 - ✓ Many protocols for security have been proved to be incorrectly after some years they are designed
- Many system vulnerabilities depend on the particular implementation
- Software vulnerabilities are not completely known



Agent-Oriented Methodologies

- AO methodologies are intended to develop organizations of autonomous agents
- They offer concepts and notations for analyze and design organizations
- The agent paradigm and related notions (e.g., goal, belief, plan, etc) are used not just to develop software systems, but also to analyze the environment in which the system will operate
- AO methodologies can be particularly suited for security requirements modeling and analysis



What is an Agent?

- A person, an organization, certain kinds of software.
- An *agent* has *beliefs*, goals (*desires*), *intentions*.
- Agents are situated, autonomous, flexible, and social.
- But note: human/organizational agents can't be *prescribed*, they can only be *partially described*.
- Software agents, on the other hand, have to be completely specified during implementation.
- Beliefs correspond to (object) state, intentions constitute a run-time concept. For design-time, the interesting new concept agents have that objects don't have is...

...goals!



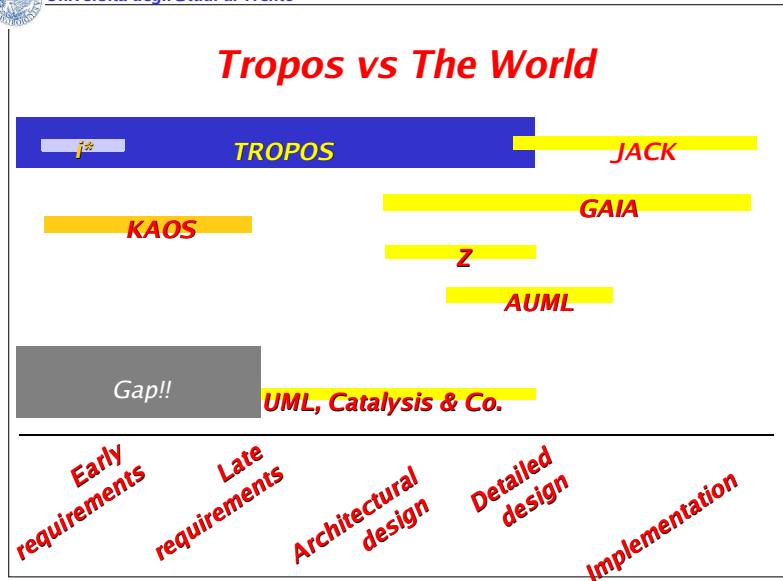
i/Tropos Methodology*

- Tropos is a research project intended to develop an agent-oriented methodology for building (agent-based) software systems
- Tropos is a joint effort among several universities and research institutes
 - ✓ <http://www.troposproject.org>
- Founded on the *i** conceptual framework
- Tailored to describe both the organization and the system itself



i - Tropos Methodology (cont)*

- Four phases of software development:
 - ✓ *Early requirements* -- identifies stakeholders and their goals. The organizational environment of a software system can be conceptualized as a set of business processes, actors and/or goals.
 - ✓ *Late requirements* -- introduce system as another actor which can accommodate some of these goals.
 - ✓ *Architectural design* -- more system actors are added and are assigned capabilities;
 - ✓ *Detailed design* -- completes the specification of system actors.



i/Tropos concepts*

- Actor
 - ✓ Intentional entity: role, position, agent (human and software)
- Goal (Soft goal)
 - ✓ Strategic interest of an actor
- Task
 - ✓ Particular course of action that can be executed in order to satisfy a goal
- Resource
 - ✓ Physical or informational entity (without intentionality)
- Social dependency (between two actors)
 - ✓ One actor depends on another to accomplish a goal, execute a task, or deliver a resource
 - ✓ Agreement between two actors



Modeling Activities

- Actor modeling, consisting on identifying and analyzing the actors of the environment, and the system's actors and agents along with their objectives
 - ✓ Early requirements phase: it focuses on modeling the application domain stakeholders
 - ✓ Late requirements phase: it focuses on the definition of the system-to-be as an actor
- Dependency modeling, consisting on identifying the goal, task and resource dependencies among actors
 - ✓ Early requirements phase: it focuses on modeling dependency relations among domain stakeholders
 - ✓ Late requirements phase: it focuses on modeling dependency relations among domain stakeholders and the system-to-be
- A graphical representation of these modeling activities is given through actor diagrams which describe the actors, their goals and the network of dependency relations among actors



Modeling Activities (II)

- Goal/Task modeling rests on the analysis of an actor's goals, conducted from the point of view of the actor, by using three basic reasoning techniques
 - ✓ Means-end analysis aims at identifying plans, resources and softgoals that provide means for achieving a goal.
 - ✓ Contribution analysis identifies goals that can contribute positively or negatively in the fulfillment of the goal
 - ✓ AND/OR refinement proceeds in decomposing a root goal into sub-goals using AND/OR decomposition
- Goal modeling is applied to requirements models aiming to refine them and to elicit new dependencies
- A graphical representation of this modeling activity is given through goal diagrams



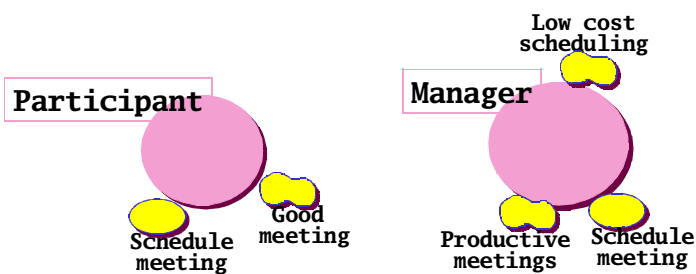
i*/Tropos - Development Process

- Initialization: Identify stakeholder actors and their goals;
- Step: For each new goal:
 - ✓ adopt it
 - ✓ delegate it to an existing actor
 - ✓ delegate it to a new actor
 - ✓ decompose it into new subgoals
- Termination condition: All initial goals have been fulfilled, assuming all actors deliver on their commitments.



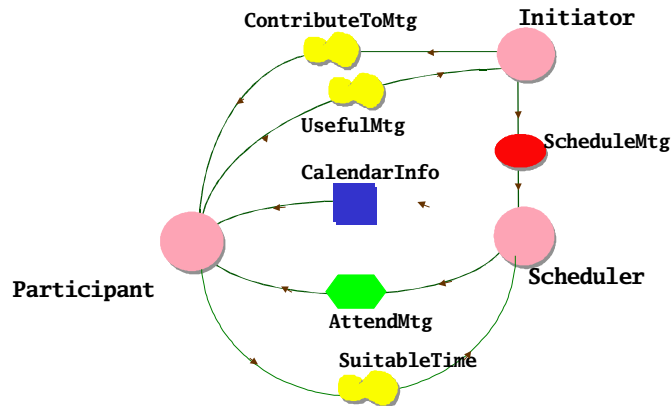
Early Requirements: Actor Modeling

A social setting consists of actors, each having *goals* (and/or *softgoals*) to be fulfilled.





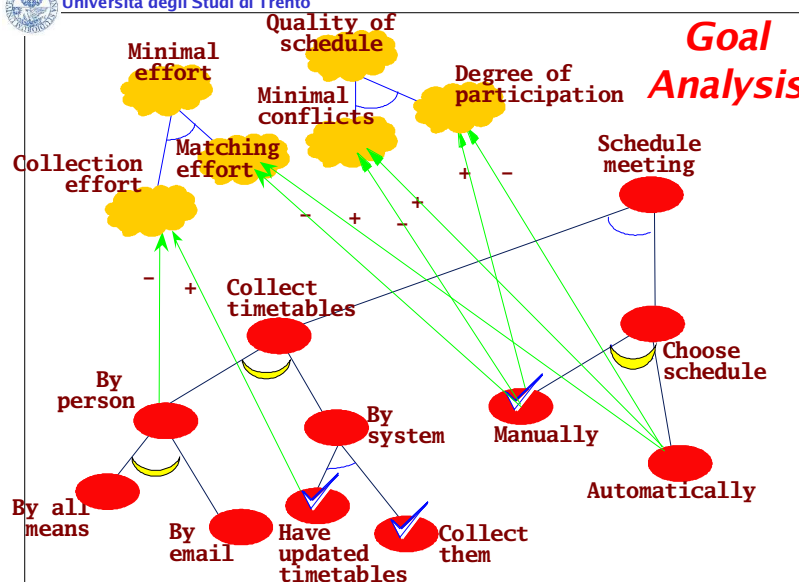
Actor Dependency Models

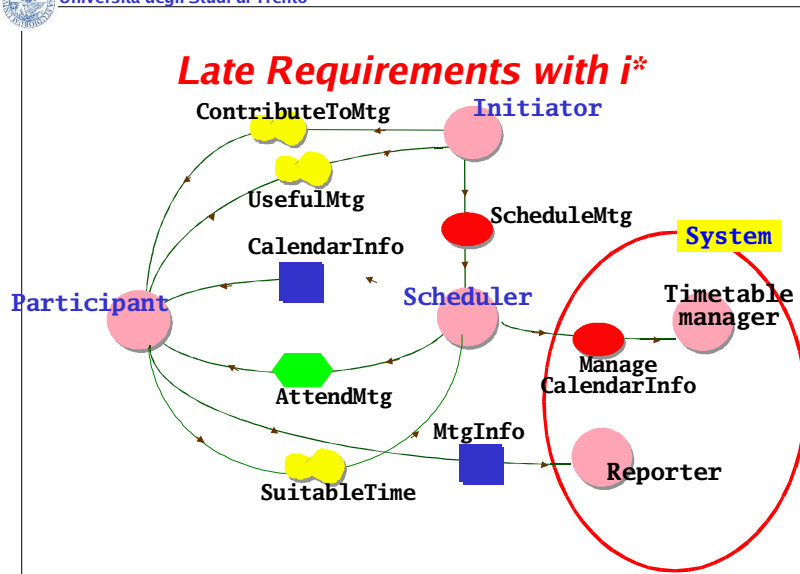


Actor dependencies are intentional: One actor **wants** something, another is **willing** and potentially **able** to deliver.



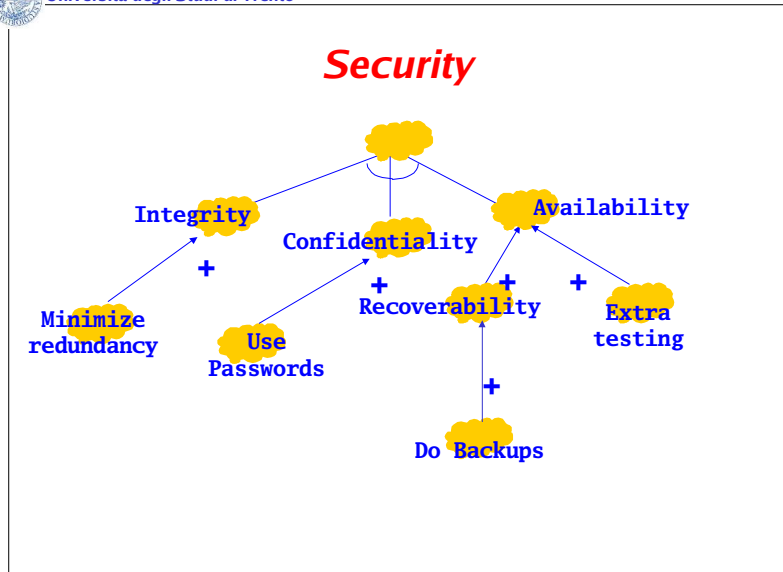
Goal Analysis





i^* /Tropos and Security

- Security is treated as non-functional requirements.
- Softgoals were introduced in [Mylopoulos92] and [Chung93] as a primitive concept for modeling non-functional requirements
- These are goals that don't have a formal definition. Consequently, softgoals don't have a clear-cut criterion as to whether they are fulfilled or not (hence their name...)
- Softgoals are *satisficed*, rather than satisfied; in other words, softgoal fulfillment is relative and "good enough", rather than absolute and optimal.
- Goal analysis is then used to verify whether security goal are satisficed



Modeling Security Concerns with i^*

- A methodological framework for security requirements analysis, which builds on a strategic actors modeling framework - i^*
- Offers facilities for analyzing threats, vulnerabilities, and countermeasures
- Relates social concerns with technology explicitly integrates security analysis with the normal requirements analysis process

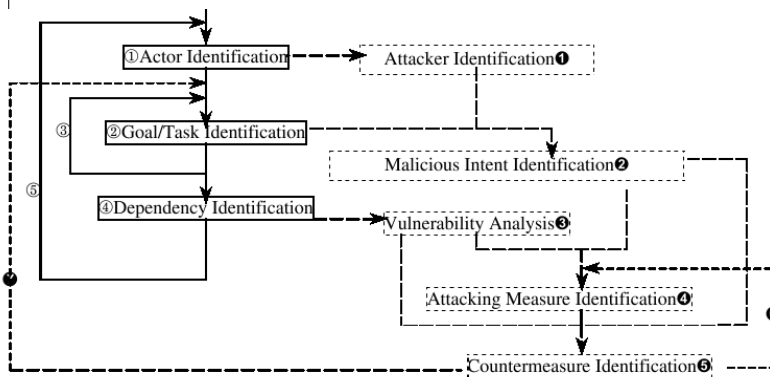


Security Requirements Analysis

- **Attacker analysis**
 - ✓ Identify potential system abusers and their malicious intents
- **Dependency Vulnerability Analysis**
 - ✓ Identify the vulnerability points in the dependency network
- **Countermeasure Analysis**
 - ✓ Make decisions on how to protect security from potential attackers and vulnerabilities
 - ✓ Identify attacks and threats
 - ✓ Introduce countermeasure
 - ✓ Countermeasure analysis is an iterative process
 - ✓ Adding protection measures may bring new vulnerabilities to the system



Requirements Elicitation Process

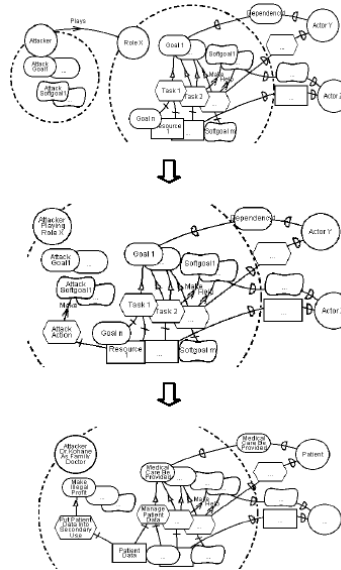


[Liu et al. 2003] Security and Privacy Requirements Analysis within a Social Setting. In Proc. of RE'03, pages 151-161.

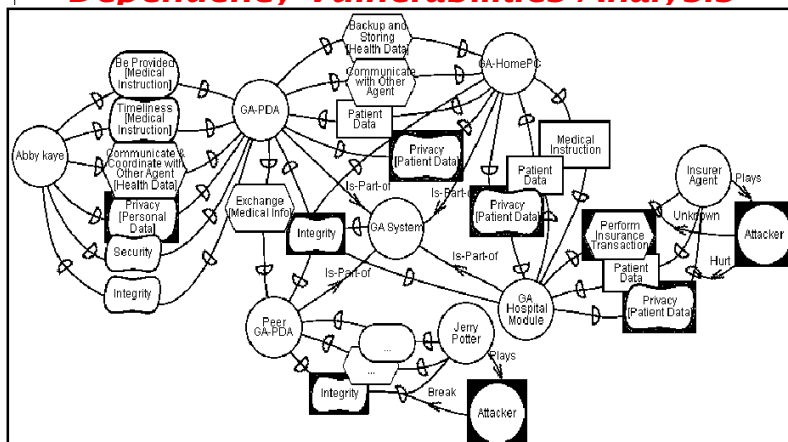
Attacker Analysis

- Actors are assumed guilty until proven innocent
- Any one of the actors identified can be a potential attacker
- The attacker inherits the intention, capabilities and social relations of the corresponding legitimate actor
- External attackers can be also considered

[Liu et al. 2003] Security and Privacy Requirements Analysis within a Social Setting. In Proc. of RE'03, pages 151-161.



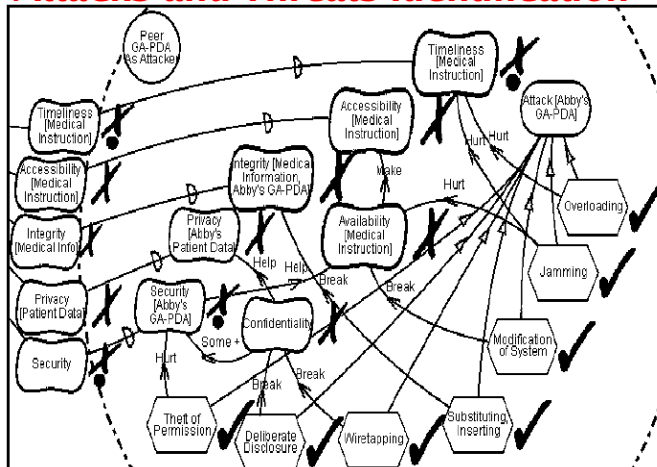
Dependency Vulnerabilities Analysis



[Liu et al. 2003] Security and Privacy Requirements Analysis within a Social Setting. In Proc. of RE'03, pages 151-161.



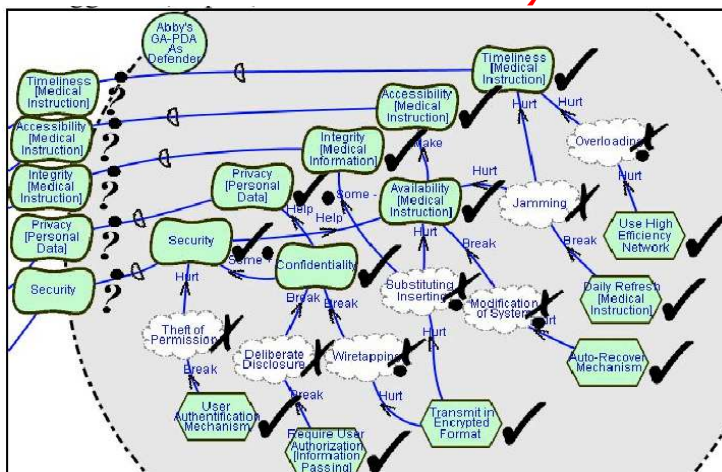
Attacks and Threats Identification



[Liu et al. 2003] Security and Privacy Requirements Analysis within a Social Setting. In Proc. of RE'03, pages 151-161.



Countermeasure Analysis



[Liu et al. 2003] Security and Privacy Requirements Analysis within a Social Setting. In Proc. of RE'03, pages 151-161.



So What?

- We have a method for identifying design features that can contribute to a security goal.
- We also have a method for evaluating the degree to which a security goal is satisfied, given a set of design decisions.
- But, formal analysis is minimal.
- Moreover, the method proposed by Chung is not specific to security!



i/Tropos vs Security*

- *i*/Tropos* has not been designed with security in mind
- Lack of the ability to capture at the same time functional and security features of an organization.
- The process of integrating security and functional requirements throughout the whole range of the development stages is quite ad hoc
- The concept of softgoal that Tropos uses to capture security requirements fails to adequately capture some constraints that security requirements often represent
 - ✓ REMARK: softgoals are goals that have no clear-cut definition and/or criteria for deciding whether they are satisfied or not
- The methodology fails to provide concepts and processes to model trust relationships



Tropos Dependency

- A dependency between two actors means that the dependee will take responsibility for fulfilling the functional goal of a depender
- Major assumption is that if you provide a service you have also the authority to decide who can use it, but...
- No way to specify or check whether the dependee is actually authorized to do so
- It can happen that an actor depends on another for a service, but the dependee is neither the owner of the service nor authorized to provide the service



NFR/i*/Tropos

- What NFR/i*/Tropos can do
 - ✓ Can support the representation of design decisions relevant to security decision
 - ✓ Can model attackers as external actors with goals and design solutions that prevent their fulfilment
 - ✓ Can support the analysis of “insiders” doing something wrong and how to prevent it
- What NFR/i*/Tropos CAN'T do:
 - ✓ model ownership and permission
 - ✓ trust is implicit in dependency



Claim

- **Ownership and permission are at the very foundation of all security concerns**
 - ✓ no ownership, no security to worry about.
 - ✓ If people didn't own human rights, privacy rights, physical property, security would be a meaningless word.
- **Permission as a complementary notion to obligation is well-accepted in Deontic Logics**
- **So, we introduce it in our modeling framework**



Moving towards a new proposal

- **Idea 1**
 - ✓ Security Requirements are social requirements
 - ✓ We need to start from a RE methodology modelling organizations
 - ✓ We need to capture the key social requirements for security
- **Idea 2**
 - ✓ We must model at the same time Functional Requirements and Security Requirements
 - ✓ So we can see the interplay of both and check one does not get in the way of the other
- **Occam's Razor**
 - ✓ Add few primitive constructs
 - ✓ Other security requirements as patterns, services, mechanisms



Secure Tropos

- Use the concepts offered by Tropos for actor, goal, task, and resource
- Make explicit who is the requester of the service, who is the legitimate owner of a service and who is able to provide a service
- Refinement of Tropos dependency
 - ✓ Trust relationship on Actor/service/Actor
 - ✓ Permission != Execution



Requiring, Ownership, and Provisioning

- Requiring
 - ✓ Identify the objectives of actors
- Ownership
 - ✓ Identify actors who are the legitimate owners of some goal, some plan, or some resource
 - ✓ The owner has full authority concerning the achievement of his goal, execution of his plan, or use of his resource
 - ✓ He can also delegate this authority to other actors
- Provisioning
 - ✓ Identify actors who have the capability to achieve some goal, execute some plan, or deliver a resource



Delegation

- **Delegation of permission**
 - ✓ Used to model formal passage of authority
 - ✓ The delegatee thinks “I have the permission to fulfill the service (but I do not need to)”
- **Delegation of execution**
 - ✓ Used to model formal passage of responsibility
 - ✓ The delegatee thinks “Now, I have to get the service fulfilled”



Trust

- **Trust is a relation between two actors representing the expectation of one actor (the trustor) about the capabilities and behavior of the other (the trustee)**
- **Trust of permission**
 - ✓ the trustor believes that the trustee will not misuse the goal, task, or resource
- **Trust of execution**
 - ✓ the trustor believes that the trustee will achieve the goal, execute the task, or deliver the resource
- **Trust is the mental counterpart of delegation**
 - ✓ Delegation is an action due to a decision, whereas trust is a mental state driving such decision



Comparing Tropos and Secure Tropos

→ Tropos Model

- ✓ Actor Properties
 - ✓ objectives
- ✓ Actor Relationships
 - ✓ dependency

• Secure Tropos Model

- ✓ Actor Properties
 - ✓ objectives
 - ✓ entitlements
 - ✓ capabilities
- ✓ Actor Relationships
 - ✓ trust of execution
 - ✓ delegation of execution
 - ✓ trust of permission
 - ✓ delegation of permission



Comparing Tropos and Secure Tropos

→ Dependency = Delegation of Execution + Trust of Execution

- ✓ if designer says A depends on B for G then A has actually delegated fulfillment of G to B and trusts that B will do it
- ✓ if one depends on X to fulfill G, X is by default authorized to do G

→ Wanting = Owning

- ✓ if designer says that A wants G, of course A is authorized to fulfill G

→ Implicit Provisioning

- ✓ When designer stops dependency chain on goal G at agent B, it means that B will take care of it

→ Trust vs Delegation

- ✓ Permit to model scenario where actors must delegate permission or execution to other actors that do not trust

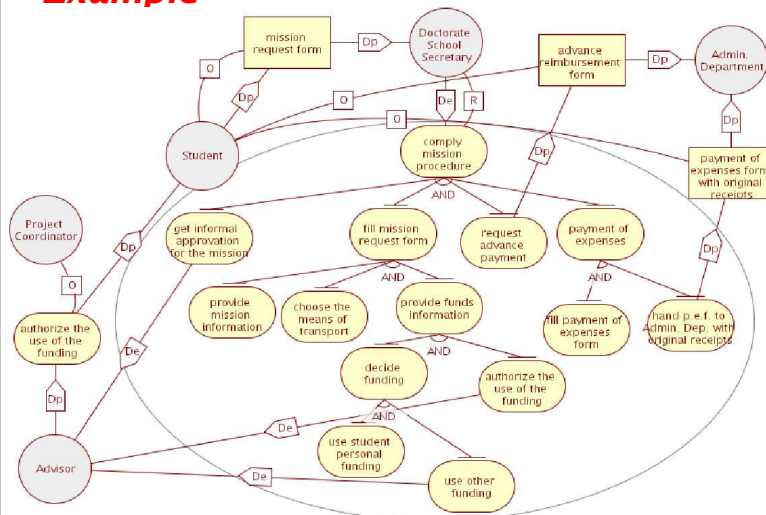


Secure Tropos -- Methodology

- Actor Diagram
 - ✓ Goals that an actors wants, provides, or owns
- Functional Requirements Model
 - ✓ Delegation of Execution
- Trust Model
 - ✓ Trust of Execution and Permission Relations
- Trust Management Implementation
 - ✓ Delegation of Permission
- Refinements by
 - ✓ Goal Decomposition within an Actor Diagram
 - ✓ Goal (Execution or Permission) Delegation to agents in a Delegation Diagram
 - ✓ Modification of Trust Relationship



Example





Underlying Formal Model

- **Formal Model**
 - ✓ Answer Set Programming (aka Datalog⁻)
 - ✓ Deduction, Satisfiability, Abduction
- **Models (secure-i* Diagrams)**
 - ✓ Extensional properties of classes (and instances)
- **Axioms**
 - ✓ Intensional properties and rules
- **Properties**
 - ✓ Specify conditions which must not be true in the model
 - ✓ Formulae that may be in true or may not be true

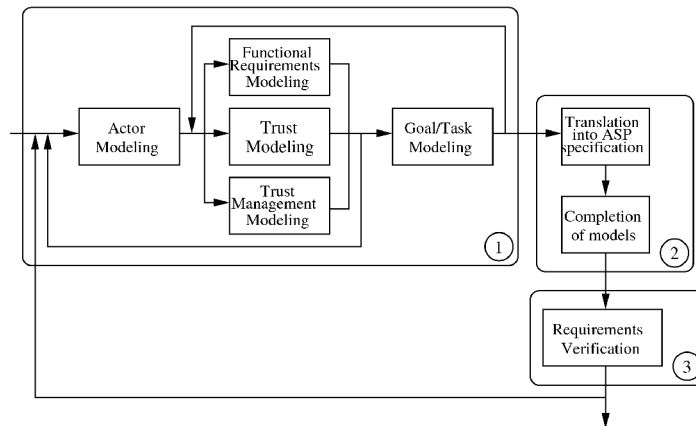


Secure Tropos for Security Services

- **Security Services as Patterns...**
 - ✓ **Authorization**
 - ✓ The owner of a service is confident that his service will not be misused
 - ✓ **Availability**
 - ✓ *Actors are confident to satisfy its objectives*
 - ✓ Actors who need to have the permission for achieving their duties have such permission
 - ✓ **Privacy**
 - ✓ Actors should have the permission on a service only if they need such a permission for achieving their duties (Need-to-know principle)



Requirements Analysis Process



Computer-Aided SRE

- Draw the graphical (Secure) Tropos models
- Diagrams (automatically) mapped into a Formal Model
 - ✓ Datalog specifications
 - ✓ Formal Tropos specification
- Check the properties on the Formal Model
 - ✓ Integration within different Datalog solvers
 - ✓ DLV System, ASSAT, C-Models, S-Models



CASE Tool Support

- **Case Study on Privacy Protection in the Enterprise**
 - ✓ CEO's goal of compliance with legal requirements and a number of refinements (eg delegation to CIO of drafting privacy policy legal documents)
- **Demos**
 - ✓ Create actor's model of goals
 - ✓ Shows various format (XML, Formal Tropos, ASP)
 - ✓ Show basic reasoning mechanisms
- **STool Demo**
 - ✓ Short (4minutes) [sttool-4min.swf](#)
 - ✓ Medium (6minutes) [sttool-6min.swf](#)
 - ✓ Long (7.5minutes) [sttool-7,30min.swf](#)



Social vs Individual

- **Tropos involves two different levels of analysis**
 - ✓ **Social level:** the structure of organizations are defined associating to every role (or position) objectives and responsibilities
 - ✓ **Individual level:** agents are not only defined with their objectives and responsibilities, but also they are associated to roles (or positions) they can play
- **In Tropos there is no explicit separation between the two levels, and it is very difficult to maintain the consistency**



Social vs Individual (II)

- It is possible that requirements are given only at individual level or at social level
- **Social => Individual**
 - ✓ The agents playing a social role “should” inherit properties and social relations of that role
 - ✓ If Alice play R1 and R1 trusts R2 and Bob plays R2 then Alice trusts Bob...
- **Useful feature to “complete” models in Computer aided RE**
 - ✓ Social relationships are always drawn in RE
 - ✓ After all they are among the system specifications
 - ✓ Designers must only draw social relationships and the reasoning system does the rest

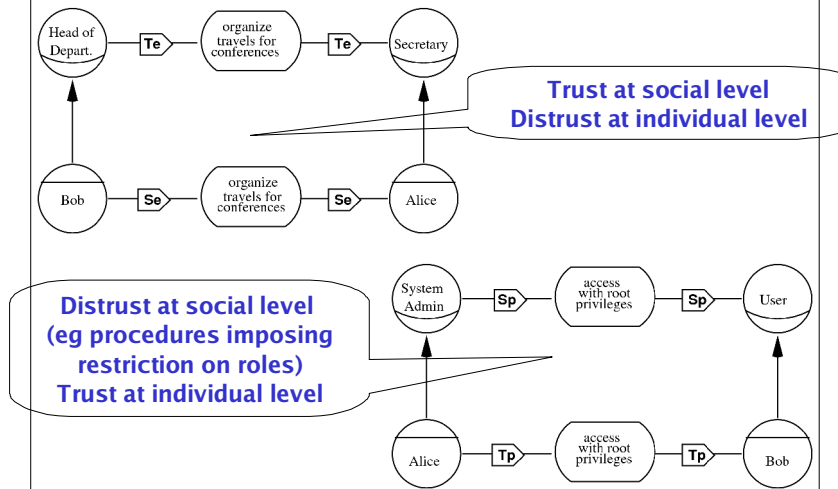


Distrust

- Need for negative authorizations to help designers in shaping the perimeter of positive trust
- Distrust is a relation between two actors representing the expectation of one actor about the incapacibilities and misbehavior of the other
- Used to identify illegitimate actors
- **Distrust as a primitive**
 - ✓ Model Trust and Distrust as independent primitive
 - ✓ Distrust as absence of trust = Close World Assumption
- **Trust Conflicts**
 - ✓ The presence of positive and negative authorization at the same time could generate conflicts on trust relationships
 - ✓ Computer Aided RE automatically detects such conflicts



Sample Conflicts



Verification Process

- Design models at both social level and individual level, independently
- Verify correctness and consistency of social level
- Map relations at social level into models at individual level
- Solve conflicts if needed
- Verify correctness and consistency of models at individual level



Monitoring

- Trust is the mental counterpart of delegation
- Delegation is an action due to a decision, whereas trust is a mental state driving such decision.
- Trust is normally necessary for delegation.
- There may be delegation without trust
 - ✓ the delegator is not free to decide (coercive delegation)
 - ✓ the delegator has no knowledge and no alternative (blind delegation)
- Delegation without trust may carry risk
- Monitoring as surrogate of trust
 - ✓ An actor (the monitor) is appointed by the delegator to monitor whether the delegatee will not misuse services and fulfill assigned obligations
 - ✓ If you do not trust somebody just monitor his work to check for error



Monitoring Conflicts

- Monitoring is just a pattern
 - ✓ Can thus be identified automatically
- Let the system find parts to be monitored
 - ✓ Modify rules for trust and distrust so that they only fire if not monitored
 - ✓ Add rules for identifying monitoring so that constraints on conflicts are not violated
 - ✓ The solver does the rest!



Privacy != Security

- Privacy is the right of individuals to determine for themselves when, how, and to what extent information about them is communicated to others.

Alan Westin - Professor of Law at Columbia Univ.

- **Contentious**

- ✓ We cannot use Software Engineering Methodologies, they have different goals (we cannot use Security Methodologies either)
- ✓ Engineering doesn't mix with civil liberties...



Real life...

- Delegation of permissions can never be as fine grained as you would need them
 - ✓ Cleaning lady has the key to open the room
 - ✓ She can empty the wastebin or steal papers from the desk.
- Real life contracts or data submissions have **purpose** tagged to permissions
 - ✓ Special power of attorney for contracts
 - ✓ Privacy statement according US, EU or national Legislation
 - ✓ You got this (permission, data, thing) to do that (action)
- If you breach trust (use for other purposes) then you can be sued, fined, etc. etc.



Privacy is Linking Permission to Purpose

- **Fact of Life**
 - ✓ We want something done
 - ✓ We give private information (or access to it) to get it done
- **If private information is used for given purpose**
 - ✓ Happy user
- **If private information is used for other purposes**
 - ✓ Consent must be sought (eg according Law)
 - ✓ Unhappy or unwilling users
- **Just map that into the Secure Tropos framework**
 - ✓ Permission on Goals is there
 - ✓ Purpose is just Delegation of Execution!



Future Work

- **More Sophisticated Reasoning**
- **Privacy Concepts**
 - ✓ Build formal theory + reasoning services
 - ✓ Relations with other formalisms (eg HippoDB)
- **Completing the Development Process**
 - ✓ Expand the model beyond early requirements
- **Ongoing Case Studies**
 - ✓ Compliance with Privacy Legislation or ISO-17799
 - ✓ John Rusnak's fraud
- **This work is partially supported by MOSTRO, SERENITY, STAMPS and SENSORIA projects**